OpenMP Raport

Zuzanna Rękawek

Kwiecień 2021

1 Specyfikacja

- Procesor: Intel Core i5-9300H
- Procesory fizyczne: 4
- Procesory logiczne: 8
- OS: Windows 10
- Typ systemu: x64

Zadanie zostało wykonane przy użyciu środowiska Microsoft Visual Studio 2019. Wszystkie programy zostały skompilowane z konfiguracją Release, a następnie uruchomione bez debbugowania.

2 Przetwarzanie sekwencyjne

Czas wykonywania obliczeń: 0.196[s]

3 Pomiary czasu

Poniższe wartości prezentują długość przetwarzania dla kolejnych wersji kodu w sekundach- podstawowej jednostce czasu układu SI.

Przykład	Wątki			Wynik
	8	4	2	
PI2	0.152412	0.168973	0.188614	niepoprawny
PI3	10.911656	7.476831	4.132476	poprawny
PI4	0.080543	0.109149	0.115842	poprawny
PI5	0.047536	0.061145	0.114769	poprawny
PI6	1.023280	0.286450	0.190930	poprawny

Tablica 1: czasy przetwarzania

Przykład	Wątki		
	8	4	2
PI2	1.29	1.16	1.04
PI3	0.02	0.03	0.05
PI4	2.43	1.80	1.69
PI5	4.12	3.21	1.71
PI6	0.19	0.68	1.02

Tablica 2: przyspieszenia

4 Omówienie działania programów

4.1 PI2

Wprowadzając derektywę *#pragma omp paralel for* sprawiono, że zmienna *i* stała się prywatną. Pozostałe zmienne są współdzielone.
Znaczącym problemem jest współdzielenie zmiennych *x* oraz *sum*- są dostępne do odczytu i zapisu. Prywatność zmiennej *x* możemy uzyskać dzięki każdorazowej deklaracji zmiennej w pętli. Współdzielenie jej jest niepoprawne- każdy wątek oblicza własną wartość.
Wielokrotny zapis i odczyt zmiennej *sum* powoduje wyścig- możliwość niesynchronizowanych dostępów. Zapis powoduje unieważnienie kopii linii zawierającej tą zmienną w innym procesorze, czego następstwem jest niepoprawny wynik końcowy.
Przewidziane przyspieszenie nie występuje w każdym przypadku, ze względu na powyższe problemy z poprawnością dostępu do zmiennych.

4.2 PI3

W tym przykładzie lokalność zmiennych nie ulega zmianie. Upewniono się, że zmienna *x* jest prywatna dla każdego wątku.
Efektem użycia dyrektywy *#pragma omp atomic* jest otrzymanie dobrego wyniku. Czas wykonywania obliczeń uległ pogorszeniu. Jest to spowodowane charakterystyką dyrektywy.
Dyrektywa *#pragma omp atomic* wymusza niepodzielność podczas operacji odczytu oraz zapisu zmiennej *sum*, będącą zmienną współdzieloną przez wszystkie wątki, które są wykonywane w sposób sekwencyjny.
Zapewnia również synchronizację na poziomie sprzętowym, czego konsekwencją jest unieważnienie linii pamięci na wszystkich innych procesorach oraz pamięci operacyjnej, które zawierają tą zmienną. Synchronizacja wątków jest realizowana za pomocą zakładanej blokady.
W celu zapewnienia atomowości uaktualnienia zmiennej współdzielonej w systemie, można skorzystać z dyrektywy *#pragma omp critical*.

4.3 PI4

W celu unieknięcia wieloktornego zapisu i odczytu w pętli zmiennej *sum* używając dyrektywy *#pragma omp atomic*, wprowadzono zmienną prywatne *sum*1, przechowującą sumy dla poszczególnych wątków.
Dyrektywa *#pragma omp atomic* jest używana poza pętlą, przy końcowym zapisie do zmiennej. Każdy wątek wykonuje tę operację tylko raz na sam koniec.
Zmienna *sum* jest współdzielona przez wiele wątków. Lokalność innych zmiennych nie uległa zmianie.

4.4 PI5

Wprowadzenie klauzuli $reduction(+ : sum)$ dodanej do $\#pragma\ omp\ for$ zapewnia prywatność zmiennej *sum*, mimo jej deklaracji poza pętlą- za każdym razem tworzona nowa, prywatna zmienna. Reszta lokalności niezmienona.
Pisząc $reduction(+ : sum)$ informujemy kompilator, że do zmiennej *sum* będziemy dodawać kolejno wyliczone wartości właśnie tej zmiennej. Kompilator utworzy odpowiednią liczbę prywatnych kopii zmiennej dla każdego wątku i rozdzieli iteracje pomiędzy dostępne wątki. Każdy wątek będzie operował tylko na swojej kopii zmiennej *sum*. Po wykonaniu wszystkich iteracji, wartości wyliczone przez wszystkie wątki są do siebie dodawane. Obliczona wartość zmiennej *sum* typu reduction jest dostępna poza sekcją parallel.
Zastosowanie klauzuli jest równoważne z operacjami w PI4. Czasy przetwarzania są bardzo podobne, biorąc pod uwagę ich szybkość wykonania.
Trzeba pamiętać, że obecne rozwiązanie ma swoje wady- przy większych, pod względem zajmowanej pamięci, strukturach może zdarzyć się, że obiekt nie zmieści się na pamięci poziomu pierwszego. Wątek traci czas na ubieganie się o miejsce na zapis zmiennej, co powoduje dłuższe przetwarzanie.

4.5 PI6

W tym przykładzie zastosowano tablicę jako strukturę przechowująca zmienne prywatne, wywoływane unikalnymi numerami id danego wątku.
Trzeba pamiętać o tym, że lokalność zmiennych uwzględniona w kodzie programu, będąca rozwiązaniem koncepcyjnie lokalnym, nie jest tym samym co lokalność z poziomu linii pamięci. Lokalność innych zmiennych niezmienona.
Program przetwarza się długo. Jest to spowodowane tym, iż elementy tablicy mogą być położone w tej samej linii pamięci podręcznej. Jest to przykład wystąpienia false sharingu- podczas zapisu wartości do zmiennej przez dany wątek, może nastąpić unieważnienie linii pamięci, z której korzystają inne wątki, czego następstwem może być wydłużenie czasu przetwarzania.

4.6 PI7

Eksperyment miał na celu wyznaczenie długości linii pamięci podręcznej procesora.
W celu otrzymania tej informacji, wykonujemy przetwarzanie na dwóch, iterujących się, sąsiednich elementach tablicy. Podczas przetwarzania można zaobserwować moment, w którym czas przetwarzania skraca się. Jest to spowodowane zniwelowaniem false sharingu- jeden z wątków pracuje na wcześniej wykorzystywanej linii pamięci, kiedy drugi przechodzi już na nową.
Powyższa sytuacja powtarza się cyklicznie co 8 iteracji. Mając na uwadze wielkość zmiennej typu *double* równą $8B$ można wyznaczyć długość linii pamięci podręcznej procesora, zgodną z architekturą komputera:

$$8 * 8 = 64B$$